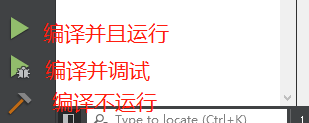
QMainWindow主要在windows下使用

QWidget常用基类。窗口类的基类



F4 跳到对应的.cpp

QPushButton b; 创建按钮，在main函数中

MyWidget w;

w.setWindowTitle("123");

//创建的窗口默认是隐藏的，需要人为显示

w.show();

QPushButton b;

b.setText("lll");//给按钮设置内容

//b.show();

/\*如果不指定父对象，对象和对象（窗口和窗口）是没有关系的

1.setParent（父对象显示，子对象自动显示）

2.通过构造杉树传参\*/

b.setParent(&w);

b.move(100,100);

b.show();

QPushButton b1(&w);//通过构造函数传参

b1.setText("bcd");

b1.show();

信号与槽函

/\*&b1 信号发出者，指针类型

\* &QPushButton::pressed 处理的信号， 发送者的类名::处理的信号

\* this 信号接收者，指针类型

\* &MyWidget::close 槽函数 信号处理函数 接受者的类名::槽函数名

\*/

QT4槽函数必须有slots关键字来修饰

connect(&b1,&QPushButton::pressed,this,&MyWidget::close);

/\*自定义槽函数，普通函数的用法

\* Qt5: 任意的成员函数，普通全局函数，静态函数

\* 槽函数需要和信号一致（参数，返回值）

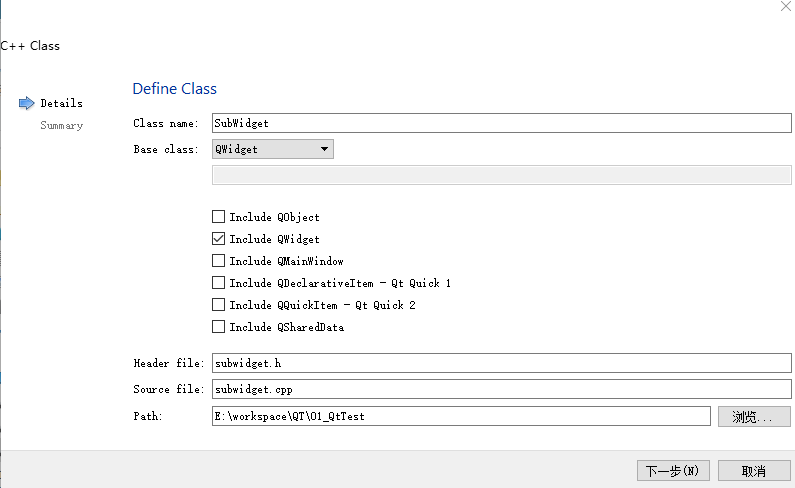
\* 由于信号都是没有返回值，所以槽函数一定没有返回值

\*/

void **mySlot**();

connect(&b1,&QPushButton::released,this,&MyWidget::mySlot);

创建子窗口，新建一个类



/\*信号必须有signals关键字来声明

\* 信号没有返回值，但可以有参数

\* 信号就是函数的声明，只需声明无需定义

\* 使用 emit mySignal();

\* 信号可以重载

\*/

void **mySignal**();

void **mySignal**(int,QString);

QDebug 和c++ cout的用法一样，用的时候qDebug()<<a<<str;

函数指针 void (\*p)(int)=fun;

信号多态处理 void (SubWidget::\*testWidget)(int,QString)=SubWidget::mySignal;

**lambda表达式 需要.pro文件添加c++11**

CONFIG+=C++11

connect(b3,&QPushButton::released,

[b3,a,b]()//a,b为只读，加关键字后可以修改[b3,a,b]() mutable

{

b3->setText("~\_~");

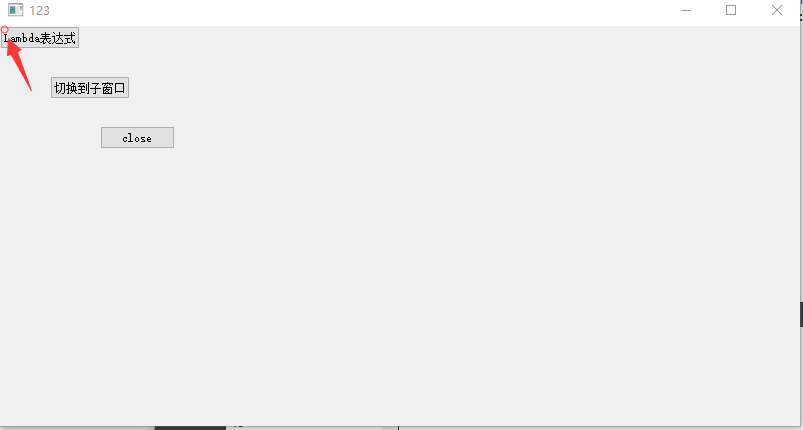
});

[=]()lambda表达式中等号表示把外部所有局部变量、类中所有成员以值传递方式传入或者课可以自己写需要传递的值[b3,a,b]

[&]()引用方式传入

**坐标系统**

对于窗口坐标原点在屏幕左上角，对于按钮坐标原点在窗口白色区域左上角



子窗口原点为相对于父窗口

直接或间接继承于QObject的类，指定父对象后，子对象如果是动态分配控件的new，不需要手动释放delete,系统会自动释放

**QMainWindow**

//添加菜单条

QMenuBar \*mBar=menuBar();

//添加菜单选项

QMenu \*pFile=mBar->addMenu("文件");

//添加菜单动作

QAction \*pAction=pFile->addAction("new");

//添加工具栏

QToolBar \*toolBar=addToolBar("tool Bar");

//工具栏添加快捷键,使用前面的指针就可以

toolBar->addAction(pAction);

QPushButton \*b=new QPushButton;

b->setText("1\_1");

toolBar->addWidget(b);

//状态栏

QStatusBar \*statBar=statusBar();

QLabel \*label=new QLabel(this);

label->setText("Normal text file");

statBar->addWidget(label);//从左向右添加

statBar->addWidget(new QLabel("3",this));

statBar->addPermanentWidget(new QLabel("6",this));//从右向左添加

QTextEdit \*qte=new QTextEdit(this);文本编辑

setCentralWidget(qte);设置中心控件

QDockWidget \*dock=new QDockWidget(this);//浮动窗口

addDockWidget(Qt::RightDockWidgetArea,dock);//主窗口增加浮动窗口

QTextEdit \*dockQte=new QTextEdit(this);//文本编辑窗口

dock->setWidget(dockQte);//浮动窗口增加文本编辑器

QAction \*pQatn=pFile->addAction("模态对话框");

connect(pQatn,&QAction::triggered,

[=]()

{

// QDialog dlg;

// dlg.show(); //显示对话框不阻塞,函数运行完变量删除，对话框一闪消失

// dlg.exec();//阻塞对话框

QDialog \*dlg=new QDialog(this);

dlg->setAttribute(Qt::WA\_DeleteOnClose);//文本框关闭后删除

dlg->show();

});